Application Performance Management in Virtualized Server Environments

Gunjan Khanna*
Dept. of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN, USA
gkhanna@purdue.edu

Kirk Beaty, Gautam Kar, Andrzej Kochut
IBM T.J. Watson Research Center,
Hawthorne, NY, USA
(kirkbeaty, gkar, akochut)@us.ibm.com

***Abstract* — As businesses have grown, so has the need to deploy I/T applications rapidly to support the expanding business processes. Often, this growth was achieved in an unplanned way: each time a new application was needed a new server along with the application software was deployed and new storage elements were purchased. In many cases this has led to what is often referred to as "server sprawl", resulting in low server utilization and high system management costs. An architectural approach that is becoming increasingly popular to address this problem is known as *server virtualization*. In this paper we introduce the concept of server consolidation using virtualization and point out associated issues that arise in the area of application performance. We show how some of these problems can be solved by monitoring key performance metrics and using the data to trigger migration of Virtual Machines within physical servers. The algorithms we present attempt to minimize the cost of migration and maintain acceptable application performance levels.**

***Index Terms:* Application performance management, virtual machine migration, virtual server.**

I. INTRODUCTION

The guiding principles of distributed computing and client-server architectures have shaped the typical I/T environment for the last three decades. Many vendors have thrived by selling elements of a distributed infrastructure, such as servers, desktops, storage and networking elements and, of course, distributed applications, such as email, CRM, etc.

As businesses have grown, so has the need to deploy I/T applications rapidly to support the expanding business processes. Often, this growth was achieved in an unplanned way: each time a new application was needed a new server along with the application software was deployed and new storage elements were purchased. In many cases this has led to what is often referred to as "server and storage sprawl", i.e., many underutilized servers, with heterogeneous storage elements. A critical problem associated with "server sprawl" is the difficulty of managing such an environment. Examples are: average use of server capacity is only 10-35 %, thus wasting resources and the bigger staff required to manage large number of heterogeneous servers, thereby increasing the total cost of ownership (TCO).

An architectural approach that is becoming increasingly popular to address this problem is known as virtualization. Virtualization occurs both at the server and the storage levels and several papers have recently been published on this topic [5], [7], [8], [12], [14]. Most of these publications deal with the design of operating system hypervisors that enable multiple virtual machines, often heterogeneous guest operating systems, to exist and operate on one physical server. In this paper we start with the concept of server level virtualization and explore how it can be used to address some of the typical management issues in a small to medium size data center.

The first system management problem we will look at in this paper is in the category of configuration management and is called server consolidation where the goal is to reduce the number of servers in a data center by grouping together multiple applications in one server. A very effective way to do this is by using the concept of server virtualization as shown in Figure 1. Each application is packaged to run on its own virtual machine, these are in turn mapped to physical machines – with storage provided by a storage area network (SAN). Details of this approach are given in Section II.

Each application in an I/T environment is usually associated with a service level agreement (SLA), which in the simplest case, consists of response time and throughput requirements. During run time, if the SLA of an application is violated, it is often because of factors such as high CPU utilization and high memory usage of the server where it is hosted. This leads to the main issue that we address in this paper: how to detect and resolve application performance problems in a virtual server based data center. Our approach is based on a novel algorithm for migrating virtual machines (VMs) within a pool of physical machines (PMs) when performance problems are detected.

In Section II, we outline a simple procedure to perform server consolidation using the concept of virtualization. Related work is presented in Section III. Section IV describes our algorithm (DMA) for dynamic migration of virtual machines and Section V presents some experimental results. We conclude the paper and outline our future research plans in Section VI.

II. BACKGROUND

The following process, very commonly used by I/T service organizations [22], provides a simple algorithm for server consolidation, as represented pictorially in Figure 1:

*The work was done while the author was a summer intern at the IBM T. J. Watson Research Center.

1-4244-0143-7/06/$20.00 ©2006 IEEE.

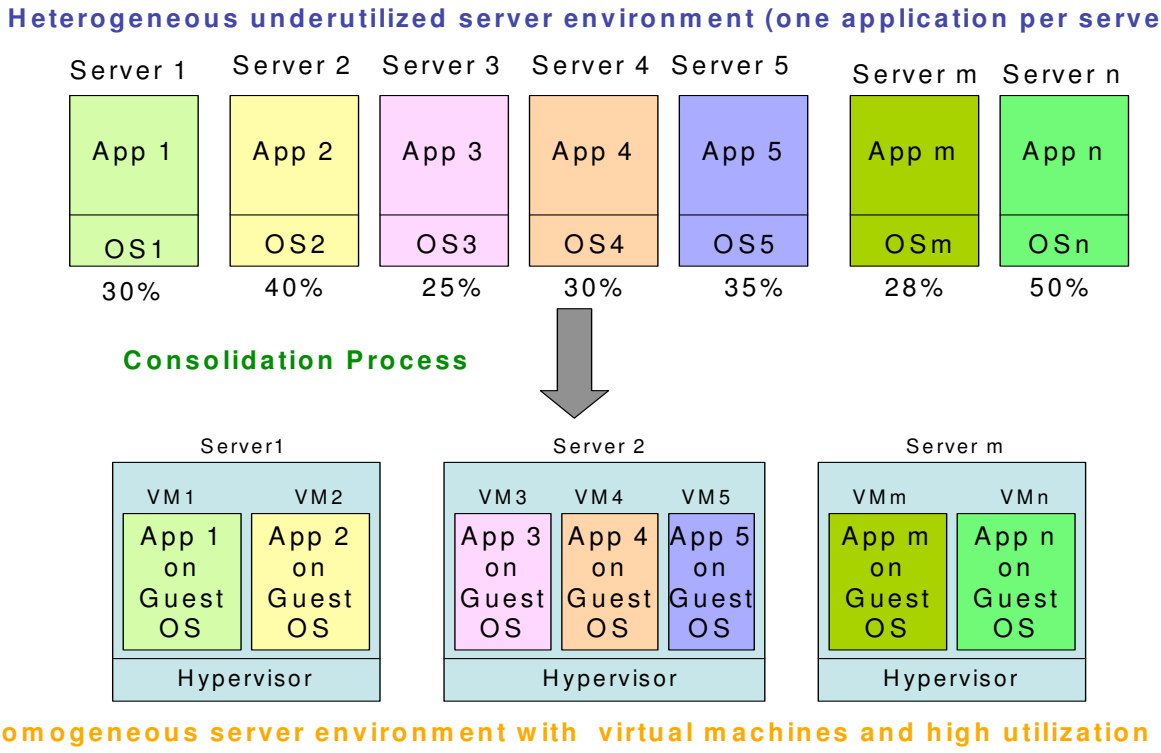

Figure 1: A Typical Virtualized I/T Environment

1) For each server to be consolidated, collect measurements that can be used to compute the average CPU usage, memory requirements, disk I/O and network bandwidth usage over a period of time (e.g., several weeks). Let us assume there are *X* servers.
2) Choose a target server type with compatible architecture, associated memory, access to shared disk storage and network communications.
3) Take each of the *X* servers, one at a time, and construct a virtual machine image of it. For instance, if server 1 is an email application on Windows, create a Windows virtual machine (e.g., using VMWare [5], [8]). The resource requirements of the virtual machine will be approximately the same as the original server which is being virtualized. At this step we will have *X* virtual machines.
4) Map the first virtual machine to the first server selected in step 2. Map the second virtual machine to the same server if it can accommodate the resource requirements. If not, introduce a new physical machine (PM) and map the VM to this new machine. Continue this step until each of the VMs has been mapped to a PM, introducing a new PM when required.
5) The set of PMs, at the end of step 4, each with possibly multiple associated VMs, comprise the new, consolidated server farm. Readers will surely associate this process with static bin packing techniques, which yields a sub-optimal mapping of VMs to physical servers.

In this paper we start with such a mapping of VMs to physical servers and propose techniques that will provide two system management functions:

a) Observe the performance of the key metrics of the operational virtualized systems (VMs) and as necessary (because of increased workload) or periodically (to optimize the consolidation when demand is less) move the virtual machines to maximize the performance; doing so while using the minimum number of physical servers.
b) If any of the VMs report an SLA violation (e.g., high response time) perform dynamic re-allocation of VM(s) from the hosting PM to another physical machine such that the SLA is restored.

III. Related Work

Related work is considered in two parts: the first looks at schemes for virtualization and server management, the second at the relevant algorithmic approaches of packing objects into bins, i.e., *bin packing*.

Virtualization has provided a new dimension for today's systems. Classic virtual machine managers (VMM) like VM/370 [20] have existed for quite some time but VMWare [8] and dynamic logical partitioning (DLPARs) [19] have provided an impetus for using virtual machines in new ways. There have been efforts to build open source virtual machine managers like Xen [6] which provides an open source framework to build custom solutions. In addition, virtualization has provided new avenues for research like Trusted Virtual Domains [23] and Grid Computing using VMs [12]. Virtualization at the application level is well addressed by products from Meiosys [18].

VMWare ESX server (hypervisor) runs on the bare hardware and provides ability to create VMs and move them from one PM to another using VMotion [8]. It requires the PM to have shared storage such as SAN or NAS. The cited paper [5] provides an overview of the memory management scheme employed in the ESX server. VMWare has the Virtual Center which provides a management interface to the virtual farm. Although some metrics are provided by the

Virtual Center, they are not fine-grained and require extensive human interaction for use in management. Resource management of these virtual machines still needs to be addressed in a cost effective manner whether in a virtual server farm or in a grid computing environment as pointed out in [12]. We provide a solution to this problem through a dynamic management infrastructure to manage the virtual machine while adhering to the SLA. A wide variety of approaches exist in the literature to perform load management, but most of these approaches concentrate on how to re-direct incoming customer requests to provide a balanced workload distribution [1] (identical to a front-end *sprayer* which selectively directs incoming requests to servers). An example is Websphere XD [4], which uses a dynamic algorithm to manage workload and allocate customer requests. The ODR component of XD is responsible for load management and for efficiently directing customer requests to back-end replicas, similar to a sprayer. Use of replication to tackle high workload and provide fair allocation with fault-tolerance has been a common practice, but for a virtual farm where each server is running a different application, it is not applicable.

The problem of allocating virtual machines to physical machines falls in the category of *vector-packing* problems in theoretical computer science (see survey [16]). It is known that finding optimal solutions to vector-packing (or its super-set *Bin-packing* and *class-constrained* packing problems) is NP-hard. Several authors including [9], [11] have proposed polynomial time approximate solutions (PTAS) to these problems with a low approximation ratio. Cited paper [10] gives an algorithm for a restricted job allocation problem with minimum migration constraint, but the problem does not allow for multiple jobs being assigned to a single machine. It is similar to the sprayer approach, developing a system which sits at the front end and makes decisions as to where to forward incoming requests. These approaches also assume that the size of the vectors and bins are fixed, i.e., deterministic values are considered. In a virtual server environment, a VM's utilization may change, thus making static allocation techniques unfit, and, instead, requiring accurate modeling and dynamic re-allocation. In order to precisely model the changing workload, authors in [13] propose stochastic load balancing in which probabilistic bounds on the resources are provided. Stochastic or deterministic packing solutions have largely looked at a static initial allocation which is close to the optimal.

It is, however, significantly more challenging to design a dynamic re-allocation mechanism which performs allocation of VMs (vectors) at discrete time steps making the system self-adjusting to workload, without violating the SLA. Also, it is important to note that the problem of minimizing migrations among bins (VMs to PMs in our case) during re-allocation is still an open research problem. Specifically, in this paper we address the issue of dynamic re-allocation of VMs, minimizing the migration cost, where cost is defined in terms of metrics, such as CPU and memory usage.

IV. Problem Formulation

We start with the environment described in the previous section, namely, an I/T environment consisting of a set of physical servers, each of which hosts one or more virtual machines (VM). In the interest of simplification of presentation, we assume that the physical server environment is homogeneous. Heterogeneous environments where PMs may have different resource capacities, e.g., CPU, memory, etc. can be handled by appropriate scaling of the migration cost matrix. Scaling is a widely used approach (see [11], [16]) to simplify the problem formulation bringing no change to the proposed solution (or algorithm). Typically each virtual machine implements one customer application. Due to workload changes, resources used by the VMs, (CPU, memory, disk and Network I/O) will vary, possibly leading to SLA violations. The objective of our research is to design algorithms that will be able to resolve SLA violations by reallocating VMs to PMs, as needed. Metrics representing CPU and memory utilization, disk usage, etc., are collected from both the VMs and the PMs hosting them using standard resource monitoring modules. Thus, from a resource usage viewpoint, each VM can be represented as a d-dimensional vector where each dimension represents one of the monitored resources. We model resource utilization of a virtual machine VM_i as a random process represented by a d-dimensional utilization vector ($U_i(t)$) at time t. For a physical machine PM_k the combined system utilization is represented by $L_k(t)$. Each physical machine, say PM_j, has a fixed capacity C_j in d-dimensional space. Assume that there are a total of n VMs which reside on m physical machines. The number of PMs (m) may change dynamically, as the algorithm proceeds, to meet the increasing workload requirements. At the initial state (t=0) the system starts with some predefined allocation (through server consolidation as outlined in Section II). As the state of the VMs change (due to changes in utilization), it causes utilization to exceed thresholds in the pre-defined allocation, leading to possible SLA violations. We propose a dynamic re-allocation of these stochastic vectors ($U_i(t)$) on the PMs to meet the required SLA. This dynamic algorithm runs at discrete time instances t_0, $t_1,\ldots,t_k\ldots$ to perform re-allocation when triggered via a resource threshold violation alert. In our model we assume a mapping of SLA to system resource utilization and hence thresholds are placed on utilization, exceeding which, triggers the re-allocation procedure. Below, we explain the nature of the inputs to the algorithm and the objective function that we attempt to optimize.

The input includes a function which maps the individual resource utilization to the combined utilization of the physical machine, i.e., $L_k(t) = f(U_1(t), U_2(t)..)$ for all VM*s* located on machine PM_k. The combined utilization is usually considered as a vector sum in traditional *vector-packing* literature but it is not generally true for several shared system resources, like SAN and CPU, because of the overhead associated with resource sharing among VMs. The latency of SAN access grows non-linearly *w. r. t.* the applied load. If we look at the average response time $R_{avg}(t)$ for all the VMs on the same PM, then it grows non-linearly as a function of the load on the physical machine (Figure 2). Let VM_j*'s* resource utilization at time t be denoted by the vector:

$$U_j(t) = [u_{j1}(t), u_{j2}(t).... u_{jd}(t)]$$

We assume that a VM's resource utilization for a specific resource is equivalent to the fraction of that resource used by

this VM on the associated PM. If A denotes the set of VMs allocated to a physical machine PM_k then the load on PM_k in the i^{th} dimension (i.e. the i^{th} resource) is given by:

$$L_i(t) = \sum_{j \in A} u_{ji}(t)$$

In general, it is hard to relate an SLA parameter, e.g. response time of a customer application, quantitatively to the utilization of the ith resource. The equation (1) approximates the non-linear behavior of the response time $R_{avg}(t)$ as it relates to the load in the i^{th} dimension on the physical machine. The n_i is the *knee* of the system beyond which the response time rises exponentially and approaches infinity asymptotically. The variable k is a tuning parameter to adjust the initial slope of the graph. Authors in [1] use a similar function for customer utility associated with a given allocation of resources to a specific customer. Their function yields a linear increase below the knee. In real systems, in order to model the graph depicted in Figure 2, we need an asymptotic increase as utilization moves close to 1, which is yielded by equation (1).

$$R_{avg}(t) = \frac{1}{2}[(L_i(t) - n_i) + \frac{\sqrt{(L_i(t) - n_i)^2 + k}}{1 - L_i(t)}] \quad (1)$$

Equation (1) is a hyperbola which closely approximates the graph in Figure 2. One can obtain similar equations for multiple dimensions. To meet the SLA requirements in terms of response time, a system should (preferably) operate below the *knee*. In each resource dimension the *knee* could occur at different points, hence the set of *knees* can be represented as a vector. This serves as a threshold vector for triggering incremental re-allocation to lower the utilizations. The utilizations are constantly monitored and the algorithm ensures, through dynamic re-allocation of VMs to physical servers, that they stay below the threshold (*knee*). Since the $L_i(t)$ are modeled as random processes, checking for a threshold violation would be done as a probabilistic guarantee $\{P(L_i(t) < n_i) > \xi\}$ which means that with probability ξ the utilization would remain below n_i; this forms a constraint.

The re-allocation procedure must consider the costs associated with performing the migration of VMs. These VMs are logical servers and may be serving real time requests. Therefore, any delay resulting from the migration needs to be considered as a cost. Use of a cost function also helps in designing an algorithm which is stable and does not cause frequent migration of machines. Let the cost of migration of one unit vector in d-dimension be denoted by the row vector M_c. It consists of migration cost coefficients for each dimension. These cost coefficients depend on the implementation of the virtual server migration. In this model we assume that the coefficient of M_c remains the same for all migrations. Thus the cost of migration of VM_j is given by $M_c.U_j(t)$. The cost of bringing in a new PM during migration is denoted by NB_c which is assumed to be orders of magnitude larger than migration cost M_c. In short, this is because, introduction of a new machine incurs hardware, software, and provisioning costs. Let matrix $R(t)$ denote the *residual capacity* of the system:

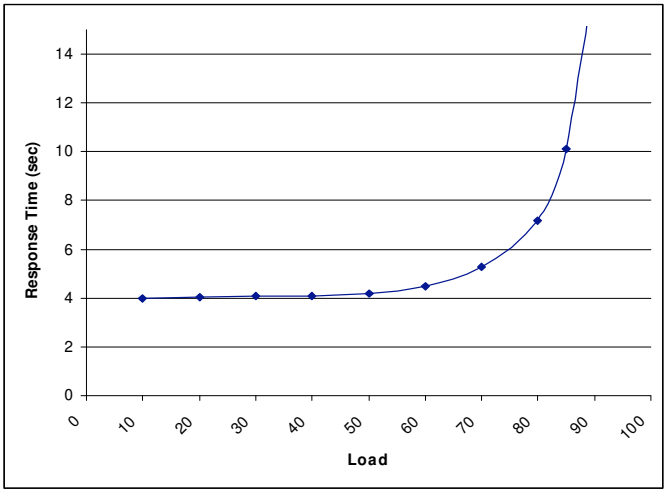

Figure 2: Response time VS Applied Load on a system dimension

$$R(t) = [r_1(t), r_2(t), r_3(t).....r_m(t)]$$

where $r_i(t)$ is the residual capacity vector of the i^{th} physical machine at time t.

Residual capacity for a resource, such as CPU or memory, in a given machine denotes the unused portion of that resource that could be allocated to an incoming VM. In order to keep the response time within acceptable bounds it is desirable that the physical machine's utilization be below the threshold (*knee)*. In some cases, such as batch applications, throughput rather than response time is the more critical SLA parameter. In such situations, thresholds can be set appropriately (from Figure 2) by specifying a higher value for acceptable response time. We would like to achieve maximum possible utilization for a given set of machines and avoid adding new physical machines unless necessary.

The aim is to achieve a new allocation of the VMs, given a previous allocation, which minimizes the cost of migration and provides the same throughput. System performance is monitored consistently for a violation of an SLA, and re-allocation is triggered when a violation occurs and is performed at discrete time instances t_1, t_2... t_k. System monitoring for system metrics can be performed using standard monitoring tools like IBM Director [24]. Because of the costs associated with migration and the use of new physical machines, it is implied that the residual capacity of a machine should be as low as possible and migrations should be minimized, thus bringing the new solution close to the previous one. It is important to note that low values of $r_i(t)$ might not be sufficient to accommodate an incoming VM during migration. Thus the goal of our algorithm is to keep the *variance of the vector R(t)* as high as possible. We will illustrate the concept behind this principle, using an example.

Let each of the $c_i(t)$ be simply CPU utilizations ($0 \leq c_i(t) \leq 100$) of the i^{th} machine. Consider the following two vectors $C(t)$: [40, 50, 30] and [90, 0, 30]. The latter vector has a higher variance of the residual vector ([10, 100, 70]) with less number of machines having high utilization. Thus, this state is more likely to be able to accommodate a migrating VM, or, in some cases a new VM, when a new customer application is introduced. Alternatively, since PM_2's resource usage, represented by the second number, is 0, it can effectively be removed. This provides us with one of the

members of the objective function that we have formulated i.e. maximizing the variance of the residual vector.

When a resource threshold is violated and the migration algorithm is set in motion, there are three decisions which it needs to make, namely:

a. Which physical machine (PM) to remove a VM (i.e., migrate) from?
b. Which VM to migrate from the chosen PM (from step 1)?
c. Which new PM to migrate the chosen VM (from step 2) to?

Since thresholds are set on the utilization at each physical machine, violation of a threshold triggers the algorithm to determine which (one or more) of the VMs from the physical machine (at which the violation took place) needs to be migrated to another physical machine.

More formally, let X_t be an n x m allocation matrix containing allocation variables x_{ij} equal to 1 if virtual machine i is allocated to physical machine j. Given an allocation at time t denoted by $\underline{X_t}$ we want to compute another allocation of machines at time $t + \Delta$ *i.e.* $X_{t+\Delta}$. The migrations performed by re-allocation is given by the migration matrix Z_M (n x *1*), obtained from the difference $X_{t+\Delta} - X_t$ and setting the rows with positive difference to be 1. The expected migration cost incurred by the new allocation is given by the scalar value:

$$E\,[M_c \cdot U(t)^T \cdot Z_M]$$

The problem in its most general form can be represented as follows:

$$\text{Max } \{ w_1 Var(R(t)) - w_2 E[M_c \cdot U(t)^T \cdot Z_M] - w_3 n.NB_c \}$$

$$P(\sum_{i=1}^{n} u_{ik}.x_{ij} < n_{jk}) > \xi \;; 1 \le j \le m \qquad (2)$$

where n is the number of new physical machines brought in to accommodate the migrating VM if necessary. For a matrix *M, Var(M)* is defined as an L2 norm of the variance vector obtained by computing sample variances of each row. For example if M_1 is a sample row with values [10 10 20], then variance of this row is 33.33. For a *n x m* matrix we first obtain a variance vector (say *A*) of size *n x 1* such that element *i* of the vector *A* is a sample variance of the values in the i^{th} row of M. Finally a L2 norm of the vector *A* gives the required scalar. Equation (2) expresses the SLA constraint which forces a physical machine's total utilization in each dimension i.e. $\sum_{i=1}^{n} u_{ik} \cdot x_{ij}$, to stay below the knee (n_{jk}). Here ξ is the probability confidence with which the utilization is below the input capacity knee n_{jk}. This equation is true for all physical machines. Thus the optimization function in this formulation consists of costs/gains associated with each of the previously discussed metrics. The maximization function can be divided into three components, each with a configurable coefficient w_i. The first sub-component reflects the gain because of *Var*(R(t)) which reflects how close the allocation is. The second term is the migration cost compared to the previous allocation. The last term is the cost incurred because of adding n new servers. Each of w_i represent the amount of weight the sub-component has on the objective function, in other words these weights are used to normalize each component to an equal scale. These can be specified by the system administrator and be fine tuned.

Weights also reflect the importance of each cost/gain function. For example, in a system where it is relatively cheaper to perform migration of VMs across the physical servers and more expensive to add a new PM, w_2 would be much lower as compared to w_3. If an administrator would like a fair utilization across physical machines and would not like to reclaim a physical machine when utilization wanes, then s/he can reduce the weight w_1. The constraint in Equation (2) represents the amount of load which each PM can hold without going over the threshold n_{jk}) and hence not violating the SLA.

A. Design Challenges

The general scenario of allocating VMs to physical machines is conceptually close to the classical *vector-packing* problem [11] making it NP-hard. As evident from the problem formulation, the number of migrations needs to be minimized and since the number of physical machines is not fixed, the use of techniques of relaxation to Linear Programming is not suitable. A solution involving LP would require re-solving the LP at each discrete time instance and a new solution might require a whole new re-allocation leading to a high migration cost. Solutions aiming to minimize the number of moves across physical machines (analogous to *bin(s)*) is still an open research problem. The solution approach must handle dynamic inclusion and exclusion of physical machines to satisfy the constraints. Cost to the customers can be calculated on the basis of usage of these PMs, providing greater flexibility at the user level.

The general problem, as represented by Equation 2, is NP-hard. In a practical setting, such as a consolidated server environment which was introduced in Section II, we would like to implement a heuristic (PTAS) algorithm that can be executed online to address SLA problems arising from over utilization of resources. In the section below we present such an algorithm which outlines actions that can be performed to optimize each of the components separately.

B. Algorithm

Assume that PM_1, PM_2….PM_m are the physical machines and VM_{ij} is the j^{th} virtual machine on PM_i. An initial allocation is already provided and the proposed dynamic management algorithm (DMA) focuses on the dynamic part. For each VM and its host physical machine the utilizations are monitored. Here utilization consists of the observed metrics like CPU, memory, etc. For each physical machine PM_i, we maintain a list of all the virtual machines allocated to it in non-decreasing utilization order, i.e. the first virtual machine, $VM_{i1,}$ has the lowest utilization. Since migration cost is directly calculated based on the utilization, another way to look at the order is "*Virtual Machines are ordered according to their migration costs within each Physical Machine*". For each physical machine we calculate and store its residual capacity $r_i(t)$. Additionally, we maintain the list of residual capacities in non-decreasing order of l^2 norm (i.e., magnitude of the vector). Without loss of generality we can represent the VMs as shown in Figure 3.The constraints, as indicated in equation (2) are constantly monitored and any violation of these constraints triggers the algorithm to

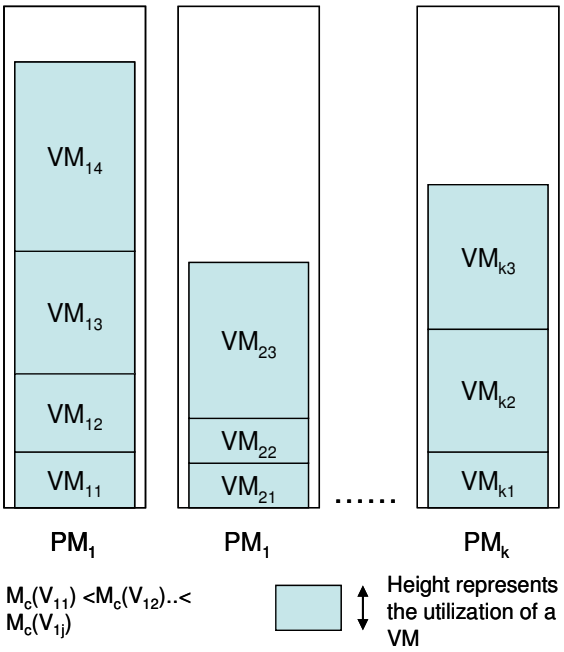

Figure 3: The VMs on each PM are ordered with respect to their Migration Costs.

perform re-allocation. Because the resource variations are not predictable, the times at which the algorithm runs is not pre-determined. Since our problem settings dictate minimum residual space, we keep lower bounds on utilization as well.

An instance of the utilization falling below a low mark for one or more physical machines, would trigger the same algorithm, but for garbage collection, i.e., reclaiming the physical machine (emptying it of VMs). Assume for physical machine PM_k the constraints are violated. Hence a VM from the physical machine PM_k must be migrated. We use the *residual capacity list* to decide the destination physical machine.

We select the VM from PM_k with the lowest utilization (i.e., the least migration cost) and move it to a physical machine which has the least residual capacity big enough to hold this VM.The process of choosing the destination physical machine is done by searching through the ordered residual capacity list (requires $O(log(m)$ time). After moving VM_{k1} the residual capacities are re-calculated and the list is re-ordered. Moving VM_{k1} might not yet satisfy the SLA constraints on PM_k, so we repeat this process for the next lowest utilization VM i.e. VM_{k2} until we satisfy the constraints. Since the destination PM (say PM_j) is chosen only if it has enough residual capacity to hold the VM (VM_{k1}), allocating VM_{k1} to PM_j does not violate the SLA constraints on that machine. In case the algorithm is unable to find a physical machine with big enough residual capacity to hold this VM, then it instantiates a new physical machine and allocates the VM to that machine. As a pre-condition to performing the migration, we compare the sum of residual capacities with the (extra needed) utilization of physical machine PM_k. If residual capacities are not enough to meet the required extra need of machine PM_k then we introduce a new physical machine and re-check. This process addresses the design constraint of having the ability to add new physical machines as required. It also might happen that the utilization falls and there is a possibility of re-claiming a physical machine. Below we describe how the algorithm reduces the number of physical machines by maximizing the variance of residual capacity if the PMs are under-utilized.

Select the virtual machine with the lowest utilization across all the PMs, which can be done in $O(m)$ time by constructing a heap, i.e. VM_{11}, VM_{21}, VM_{31}….VM_{m1}. It is important to note that once the heap is constructed, in all subsequent rounds of the algorithm only $O(1)$ time would be required since *ExtractMin* in a Min-heap is a constant order operation. We move this VM (say VM_{k1}) to another physical machine which has the minimum residual capacity just big enough to hold this machine such that it increases the *Variance*(R), where R is the residual capacity vector. We only move a VM if moving it causes the *Var*(R) to increase, otherwise we choose the next smallest VM. We repeat this step until *Var*(R) starts decreasing; this defines a termination condition. Also when there is no residual space which can fit a chosen VM, the algorithm terminates. In every iteration we pack the VMs as closely as possible thus trying to minimize the number of physical machines used. If a physical machine ends up having no VMs left on it then it can be removed by means of garbage collection.

C. Important Features of the algorithm

We provide a PTAS which can be used to perform *online* dynamic management.

It builds on an existing allocation and when invoked, because of an SLA violation, tries to minimize the number of migrations.

It minimizes the migration cost by choosing the VM with minimum utilization.

It provides mechanism to add and remove physical machines thus providing dynamic resource management while satisfying the SLA requirements i.e. meeting the response time and maintaining the throughput of the virtual servers.

V. Experiments and Results

A. Test-bed

We have used an IBM BladeCenter environment with blades as our physical machines. VMWare ESX server (hypervisor) is deployed on three bare HS-20 Blades (Intel architecture). We create VMs on top of the hypervisors.

Figure 4 shows the logical topology of the experimental set-up. Blades in the IBM BladeCenter are the physical machines of the model. Each of the three blades has the VMWare ESX hypervisor installed. For each blade, the Virtual Machines (VM_i) are created on top of the ESX server giving each of them equal shares of the physical CPU. Each VM has 4 GB hard disk space and 256 MB of RAM. The BladeCenter is connected to a storage area network (SAN) which provides shared storage for all the blades. The virtual machine images are stored in the SAN. The environment is managed using IBM Director, consisting of agents and a management server.

IBM Director Agents are installed on each blade (i.e., the hypervisor) and on the VMs as well. The IBM Director Server sits on a separate machine and pulls management data from the director agents. The management data consists of metrics like CPU, Memory, I/O, etc. The IBM Director

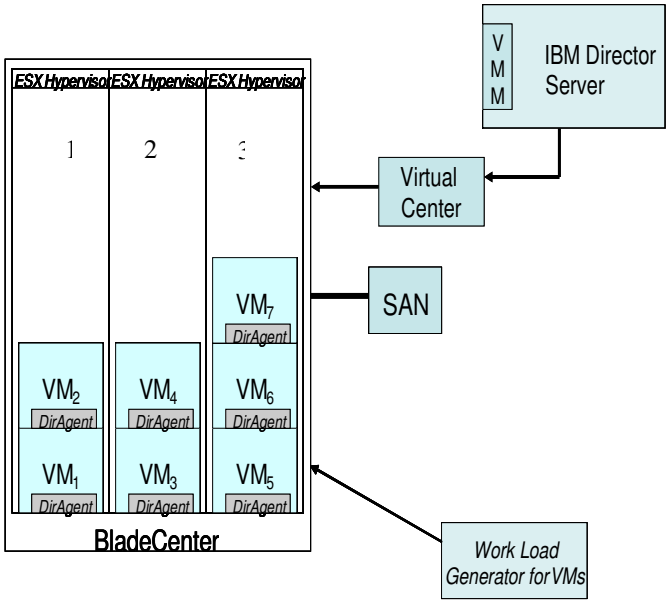

Figure 4: Test-Bed Logical Topology

Server also contains IBM's Virtual Machine Manager (VMM) server. VMM allows exporting virtual machines to the director console through interaction with VMWare's Virtual Center. VMM provides all the actions which one can perform through the Virtual Center to the Director console for e.g., migration of a VM from one blade to another blade, powering on/off a VM, etc. Migration of VMs from one blade to another is carried out by the Virtual Center through the tool called VMotion.

Referring to Figure 4, VM_1 is a Linux machine running the Lotus Domino server (IBM's messaging and collaboration server). VM_2 is a Windows machine which has IBM DB2, IBM Websphere Application Server (WAS) and the Trade3 application installed. Note VM_3 is a clone of VM_2. All the other VMs are Linux machines running little or no load. Since the blades share the disk and network, we only consider CPU and memory migration costs, i.e., the utilization vector consist of only 2 dimensions. We consider memory migration cost because *VMotion* transfers the VM's RAM (entire *hot* state) during migration process. We consider CPU cost because experimental evidence has shown that delay in migration increases as CPU load increases. We use *Websphere Workload Simulator* (WSWS) to generate workload for the Trade3 and employ *Server.Load* scripts to generate workload for the Lotus Domino server.

We create an event filter using the *simple event filter* function offered by IBM Director and associate this event filter with the system metrics (like CPU, memory). Thresholds are set in the event filter so that an event is generated whenever the threshold is exceeded. An *Event Action Plan* (EAP) is created to contain the actions which need to be executed if an event is triggered. Our algorithm, DMA (dynamic management algorithm), becomes a part of the Event Action Plan and gets executed when an associated event is triggered. SLA metrics including the response time and utilization thresholds along with the costs are an input to the DMA.

To demonstrate proof of concept, we set up a simple experiment to show how IBM Director might be used to implement the DMA defined here. We created an IBM Director based event filter to monitor the CPU of blade 1 hosting VM_1 and VM_2 (Figure 4), named as *CPU_filter*. We created an EAP for migration of one of the VMs (VM_1) to the neighboring blade 2 if the *CPU_filter* generates an event. We turned on the workload generators for VM_1 and VM_2 which causes the CPU utilization of the blade to increase. The event filter which is applied to blade 1 generates an event because CPU utilization exceeds the predefined threshold. Generation of the event triggers the associated EAP and automatically migrates VM_1 to blade 2. We used this setup to perform successful dynamic migrations of VMs between the blades by monitoring the system metrics, CPU utilization and memory usage.

B. Goodness of the Proposed Algorithm

We measure the goodness of DMA by performing extensive studies in a simulation environment because it is not feasible to obtain all the possible conditions in a test-bed. Our algorithm is used in the simulation study with utilizations of VMs provided as input.

We compare DMA to an *optimal* algorithm which enumerates all the possible permutations of the VMs, allocated to the physical machines, and finds the allocation with maximum residual variance, i.e., provides optimally packed VMs on the given PMs. We measure and compare the migration cost and residual variance of physical machines used for optimal allocation with those used by the allocation yielded by DMA. We use the residual variance instead of simply the number of physical machines because for a given number of physical machines residual variance is a good measure to decide the quality of allocation. Since the optimal algorithm (for NP-hard problem) searches over the entire solution space, it has an exponential complexity.

The initial allocation of VMs to PMs is obtained from the optimal algorithm, which is fed to the DMA. At each iteration, we randomly choose a PM and change the utilization of one of its VMs. We perform re-allocation of VMs using both of the algorithms when an SLA violation (or under-utilization) occurs. Changing a VM's utilization may or may not trigger a re-allocation, depending on whether or not the set thresholds are violated. At the end of each re-allocation, costs are calculated relative to the previous allocation, as provided by DMA. The migration cost vector, M_c, contains non-zero weights for CPU and memory, because they are the metrics which affect our test bed during migrations. Note that, in practice, M_c depends on the virtualized server environment and the details of how migration is carried out. We measure the ratio of cost of migration of DMA with that of the optimal algorithm, thus nullifying the effect of absolute numbers in vector M_c. For simplicity we assume each machine has a unit capacity in each resource dimension. In a real scenario, PMs might have varying thresholds for utilization depending upon SLA requirements.

Figure 5 shows the variation of residual variance ratio with an increasing number of virtual machines. DMA dynamically increases/decreases the actual number of physical machines that need to be used. For every run the initial value of PMs is set to 2. Each data point is averaged over a minimum of 100 re-allocations. Ideally, the ratio of the residual variances should be close to 1.

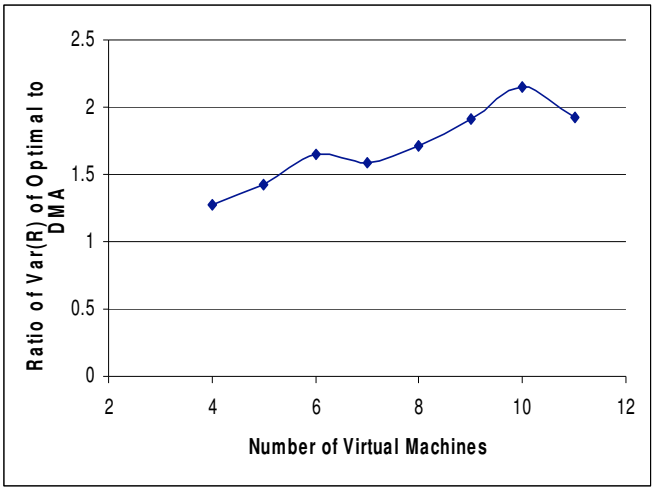

Figure 5: Comparison of Residual Variance in the Optimal VS the residual variance of our algorithm (DMA).

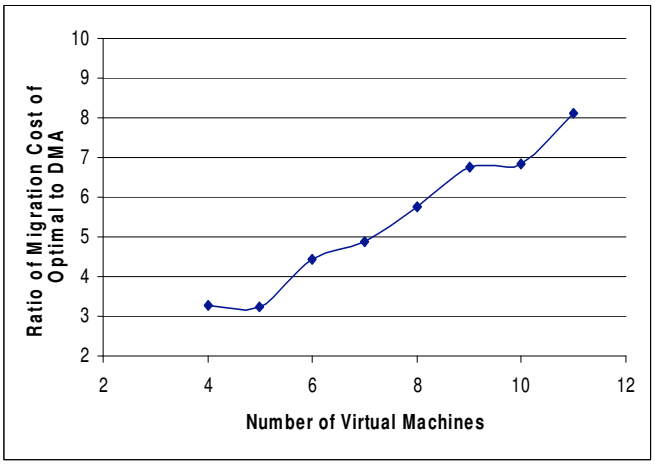

Figure 6: Ratio of cost of migration yielded by the optimal algorithm VS migration cost of the proposed algorithm (DMA) with increasing number of VMs.

The experiments show that for DMA the ratio stays between 1.3 and 2.1 for up to 11 VMs. We note that the performance of DMA degrades, as compared to the optimal, as the number of VMs in the system increases. This increasing trend in the ratio can be attributed to the fact that the optimal algorithm has greater flexibility of searching over more permutations during reallocation. Additionally, our proposed solution accounts for migration cost, which the optimal algorithm does not, further reducing the allocation choices that it has.

Figure 6 best explains the above notion by plotting the ratio of the average migration cost incurred by DMA versus that of the optimal. The optimal algorithm incurs migration costs which is 3 to 8 times more than DMA. As the number of VMs increases, the optimal algorithm performs worse because it tries to form a closely packed allocation, inducing lots of migrations, starting from the configuration offered by the prior solution.

In summary, DMA does not perform as well as the optimal bin packing technique (as implemented by an exhaustive enumeration) in terms of the residual variance, but does considerably better in terms of migration costs. Here we only compare the performance upto eleven VMs because in practice, even if there are a large number of VMs spread over a large number of physical machines, migration will generally not be allowed across the entire set of physical machines. Rather, migration will be limited within smaller clusters of physical machines, such as within a department or within a similar application group. Also, migration across LAN is neither desirable nor feasible using the current technology. Thus, we think, that the performance degradation of DMA with increasing number of VMs is not likely to be a serious drawback in real life. Being an online algorithm DMA can be deployed in any management software to help manage a virtualized environment.

VI. Conclusion

Today, many small to medium I/T environments are reducing their system management costs and total cost of ownership (TCO) by consolidating their underutilized servers into a smaller number of homogeneous servers using virtualization technologies such as VMWare, XEN, etc. In this paper we have presented a way to solve the problem of degrading application performance with changing workload in such virtualized environments. Specifically, changes in workload may increase CPU utilization or memory usage above acceptable thresholds, leading to SLA violations. We show how to detect such problems and, when they occur, how to resolve them by migrating virtual machines from one physical machine to another. We have also shown how this problem, in its most general form, is equivalent to the classical bin packing problem, and therefore can only be practically useful if solved using novel heuristics. We have proposed using migration cost and capacity residue as parameters for our algorithm. These are practically important metrics, since, in an I/T environment, one would like to minimize costs and maximize utilization. We have provided experimental results to validate the efficacy of our approach when compared to more expensive techniques involving exhaustive search for the best solution.

There are several areas that can be identified for interesting future work:

use of application workload profiles, e.g., variation of load during the day, as an input to the migration algorithms; in general we should avoid putting two virtual machines together when their workload profiles make it more likely that resource usage thresholds will be exceeded because of similar usage patterns,

predict metric threshold violations based on analysis of application profiles, leading to a proactive problem management system,

characterize migration cost more realistically in terms of application properties, for example required communication paths with other applications,

provide fault tolerance using frozen images of virtual machines; when a physical machine fails, bring in another machine quickly (in Blade Center such hardware level fault tolerance is relatively easy to implement) and configure it from the frozen VM images.

Acknowledgments

We would like to acknowledge Michael Frissora, James Norris and Charles Schulz for their assistance in obtaining the needed hardware and software under time critical constraints. We are also thankful to James Norris and Anca

Sailer for lending their expertise during the software installation phase and to Norman Bobroff for his contributions to discussions related to the technical content of this work.

REFERENCES

[1] A. Chandra, W. Gong, and P. Shenoy, "Dynamic Resource Allocation for Shared Data Centers using Online Measurements," IWQoS, 2003.

[2] A. Chandra and P. Shenoy, "Effectiveness of dynamic Resource allocation for handling Internet Flash Crowds," Technical Report TR03-37, Department of Computer Science, University of Massachusetts Amherst, November 2003.

[3] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar, "Stream-Packing: Resource Allocation in {Web} Server Farms with a QoS Guarantee," Lecture Notes in Computer Science.

[4] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi "Dynamic application placement under service and memory constraints," in Proceedings of WEA 2005, pp. 391-402.

[5] C.A. Waldspurger, "Memory resource management in VMware ESX server," Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), 2002.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," Symposium of Operating Systems Principles, 2003.

[7] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum "Optimizing the Migration of Virtual Computers", Operating System Design and Implementation, pp 377 - 390, 2002.

[8] http://www.vmware.com/

[9] H. Shachnai and T. Tamir, "Noah's Bagel-some combinatorial aspects", International Conference on FUN with algorithms (FUN), Isola d' Elba, June 1998.

[10] T. Kimbrel, B Schieber, and M. Sviridenko, "Minimizing migrations in Fair Multiprocessor scheduling of persistent Tasks", Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, 2004.

[11] C. Chekuri and S. Khanna, "On Multi dimensional Bin packing problems," in Proceedings of the 10th Symposium on Discrete Algorithms, pp 185-194, 1999.

[12] R. J. Figueredo, P. A. Dinda, and J. A. B. Fortes, "A case for Grid Computing on Virtual Machines" Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003.

[13] A. Goel and P. Indyk, "Stochastic Load Balancing and related Problems" Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999.

[14] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum, "Cellular Disco: resource management using virtual machines on shared memory multiprocessors", 17th ACM Symposium on Operating Systems designs and principles (SOSP'99), 1999.

[15] A. Awadallah and M. Rosenblum, "The vMatrix: A network of Virtual Machine Monitors for dynamic content distribution," IEEE 10th International Workshop on Future Trends in Distributed Computing Systems (IEEE FTDCS 2004), Suzhou, China, May 2004.

[16] S. Kashyap and S. Khuller, "Algorithms for Non-uniform Size data placement on Parallel Disks," Journal of Algorithms, FST&TCS 2003.

[17] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson, "Approximation Algorithms for Bin Packing: A Survey," Approximation Algorithms for NP-Hard Problems, D. Hochbaum (editor), PWS Publ., Boston (1997), pp.46-93.

[18] www.meiosys.com.

[19] http://www.research.ibm.com/journal/sj/421/jann.html.

[20] http://www.vm.ibm.com/overview/zvm52sum.html.

[21] http://www-1.ibm.com/servers/eserver/zseries/zos/bkserv.

[22] http://www-1.ibm.com/servers/eserver/iseries/scon.

[23] J. L. Griffin, T. Jaeger, R. Perez, R. Sailer, L. van Doorn, and R. Cáceres, "Trusted Virtual Domains: Toward Secure Distributed Services," Proc. of 1st IEEE Workshop on Hot Topics in System Dependability (HotDep 2005), June 2005.

[24] http://www-1.ibm.com/servers/eserver/xseries/systems_management/director_4.html.

[25] M. Dahlin, "Interpreting the State Load Information," 19th International Conference on Distributed Computing Systems, May-june, 1999.

[26] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance Management for cluster based Web-Services," IBM Technical Report.